Attribute-Sets: A Practically Motivated Enhancement to Attribute-Based Encryption

Rakesh Bobba, Himanshu Khurana and Manoj Prabhakaran

University of Illinois, Urbana-Champaign IL USA,
{rbobba,hkhurana,mmp}@illinois.edu

Abstract. In distributed systems users need to share sensitive objects with others based on the recipients' ability to satisfy a policy. Attribute-Based Encryption (ABE) is a new paradigm where such policies are specified and cryptographically enforced in the encryption algorithm itself. Ciphertext-Policy ABE (CP-ABE) is a form of ABE where policies are associated with encrypted data and attributes are associated with keys. In this work we focus on improving the flexibility of representing user attributes in keys. Specifically, we propose Ciphertext Policy Attribute Set Based Encryption (CP-ASBE) - a new form of CP-ABE - which, unlike existing CP-ABE schemes that represent user attributes as a monolithic set in keys, organizes user attributes into a recursive set based structure and allows users to impose dynamic constraints on how those attributes may be combined to satisfy a policy. We show that the proposed scheme is more versatile and supports many practical scenarios more naturally and efficiently. We provide a prototype implementation of our scheme and evaluate its performance overhead.

1 Introduction

In distributed systems users need to share sensitive objects with others based on the recipients' ability to satisfy a policy. Attribute-Based Encryption (ABE) ushers in a new paradigm where such policies are specified and cryptographically enforced in the encryption algorithm itself. Existing ABE schemes come in two complimentary forms, namely, Key-Policy ABE (KP-ABE) schemes and Ciphertext-Policy ABE (CP-ABE) schemes. In KP-ABE schemes [13, 14, 16, 18], as the name indicates, attribute policies are associated with keys and data is annotated with attributes. Only those keys associated with a policy that is satisfied by the attributes annotating the data are able to decrypt the data. In CP-ABE schemes [2, 7, 12, 15], on the other hand, attribute policies are associated with data and attributes are associated with keys. Only those keys whose associated attributes satisfy the policy associated with the data are able to decrypt it.

CP-ABE is more intuitive as it is similar to traditional access control model where data is protected with access policies and users with credentials satisfying the policy are allowed access to it. Among the various CP-ABE schemes proposed the one proposed by Bethencourt *et al.* [2], which we will hereafter refer to as BSW, is the most practical to date. It supports arbitrary strings as attributes, numerical attributes in keys and integer comparisons in policies and provides a means for periodic key refreshment. Furthermore, the authors have developed a software prototype with a friendly interface for integration in systems. However, BSW and other CP-ABE schemes are still far

from being able to support the needs of modern enterprise environments, which require considerable flexibility in specifying policies and managing user attributes as well as increased efficiency. This is in part due to the fact that keys in current CP-ABE schemes can only support user attributes that are organized logically as a single set; *i.e.*, users can use all possible combinations of attributes issued in their keys to satisfy policies. This, we observe, imposes some undesirable restrictions which are outlined below.

First, this makes it both cumbersome and tedious to capture naturally occurring "compound attributes", *i.e.*, attributes build intuitively from other (singleton) attributes, and specifying policies using those attributes. For example, attributes that combine a traditional organizational role with short-term responsibilities result in useful compound attributes; *e.g.*, 'Faculty' in 'College of Engineering' serving as 'Committee Chair' of a 'University Tenure Committee' in 'Spring2009' are all valid attributes in their own right and are likely to be used to describe users. The only way to prevent users from combining such attributes in undesirable ways when using current CP-ABE schemes is by appending the (singleton) attributes as strings; i.e., *faculty_collegeOfEngineering_committeeChair_univTenureCommittee_Spring2009*. But this approach has an undesirable consequence in that it makes it challenging to support policies that involve other combinations of singleton attributes used to build the compound attribute; *e.g.*, policies targeting "all committee chairs in Spring2009" or "faculty serving on tenure committees". This is because the underlying crypto in CP-ABE schemes can only check for equality of strings and thus cannot extract the "faculty" or "committeeChair" attributes from a compound attribute such as the one described above.

Second, CP-ABE schemes that support numerical attributes (*i.e.*, allow numerical comparisons in policies) are limited to assigning only one value to any given numerical attribute within a key. But there are many real world systems where multiple numerical value assignments for a given attribute are common; *e.g.*, students enrolled in multiple courses identified by numeric course numbers in a given semester, users with multiple accounts at a particular bank, disease codes for individual diseases and disease classes used widely in health care. Furthermore, the ability to compare across such multiple value assignments adds flexibility to policy specification. For example, consider a college student enrolled in two junior level courses, 357 and 373, and two senior level courses, 411 and 418 respectively. Without support for multiple numerical value assignments for a given attribute specifying policies to target students enrolled in senior level courses, such as "course number greater than or equal to 400 and less than 500" is tedious and cumbersome.

Our Contribution In this work we propose Ciphertext-Policy Attribute-Set Based Encryption (CP-ASBE), a form of CP-ABE, that addresses the above limitations of CP-ABE by introducing a recursive set based structure on attributes associated with user keys. Specifically CP-ASBE allows, 1) user attributes to be organized into a recursive family of sets and 2) policies that can selectively restrict decrypting users to use attributes from within a single set or allow them to combine attributes from multiple sets. Thus, by grouping user attributes into sets such that those belonging to a single set have no restrictions on how they can be combined, CP-ASBE can support compound attributes without sacrificing the flexibility to easily specify policies involv-

ing the underlying singleton attributes. Similarly, multiple numerical assignments for a given attribute can be supported by placing each assignment in a separate set.

While restricting users to use attributes from a single set during decryption can be thought of as a regular CP-ABE scheme, the challenge in constructing a CP-ASBE scheme is in selectively allowing users to combine attributes from multiple sets within a given key while still preventing collusion, *i.e.*, preventing users from combining attributes from multiple keys. We provide a construction for a CP-ASBE scheme that builds on BSW, prove its security in the generic group model and evaluate its performance through a prototype implementation.

The rest of this paper is organized as follows. Section 2 further motivates CP-ASBE. Section 3 discusses related work. In Section 4 we give some preliminaries. We present our construction and discuss its security in Section 5. In Section 6 we discuss efficiency of the scheme, give details of our prototype implementation and discuss performance. Section 7 concludes the paper and discusses future directions.

2 Motivation

The ability to group attributes into sets and to frame policies that can selectively restrict the decrypting key to use attributes belonging to the same set is a powerful feature more than one might realize initially. In this section we illustrate its versatility by solving various problems in different contexts which did not have any reasonably efficient solutions prior to this.

2.1 Supporting Compound Attributes Efficiently

While existing CP-ABE schemes offer unprecedented expressive power for addressing users, for several natural scenarios they are inadequate. We illustrate this with the following natural example and show how CP-ASBE provides a simple solution.

Consider attributes for students derived from courses they have taken. Each student has a set of attributes (Course, Year, Grade) for each course she has taken. In the following, consider a simple policy "Students who took a $300 \leq$ Course < 400 in Year ≥ 2007 and got Grade > 2." Using a CP-ABE scheme for this is challenging because, for instance, a student can take multiple courses and obtain different grades in them. The policy circuit will have to ensure that she cannot mix together attributes from different sets to circumvent the policy. We point out a few possible options of using CP-ABE, but all unrealistic or unsatisfactory. The efficiency parameters considered are the number of designed attributes given to each student, and the size of the designed policy (a circuit, with designed attributes as inputs, for enforcing the policy).

- For each course that the student has taken, let there be a single designed (boolean) attribute that she gets (e.g. cyg:373_2008_4). But the designed policy will have to (unrealistically) anticipate all such attributes that will satisfy the policy (e.g., cyg:300_2007_3 or cyg:301_2007_3 or ... or cyg:399_2010_4).
- Anticipate (again, unrealistically) all possible policies that may occur which the student's attributes will satisfy, and give her compound boolean attributes corresponding to each of these policies (e.g., cyg:373_2008_4, cyg:373_2008,

cyg:(≥300)_2008, cyg:(≥400)_2007-or-cyg:(≥300)_2008_(≥3), ...). In this case our designed policy is minimal, with just an input gate (labeled by the attribute cyg:(≥ 300,< 400)_(≥ 2007)_(> 2)) and an output gate.

– Fix an upper bound on the number of courses a student could ever take, say 50, and give all attributes indexed by a counter (e.g. Course#1, Year#1, Grade#1 etc.); then the policy will have to incorporate several cases (e.g., ($400 <$ Course#1 ≥ 300 and Year#1 ≥ 2007 and Grade#1 > 2) or ... or ($400 <$ Course#50 ≥ 300 and Year#50 ≥ 2007 and Grade#50 > 2)). This increases the policy size by a factor of 50.

If a policy can refer to more than one course, all these approaches will lead to even more inefficiency or restrictions. For example, in the third (and the most efficient) approach, if the policy refers to two courses, the blow up will be by a factor of 100 instead of 50.

We stress that these are not the only possibilities when using CP-ABE. In general, by giving more attribute keys, the circuit complexity of the policies can be reduced (the first two options above being close to the two extremes). One could achieve slightly smaller policies by adding judiciously chosen auxiliary attributes and adding some structure to values taken by these attributes (for instance, in the third option above, one can let the counter monotonically increase with the course number). However, the resulting schemes are still unrealistically inefficient in terms of policy size and/or number of keys, and *further* makes attribute revocations even less efficient.

A CP-ASBE scheme can be used to overcome these issues by assigning multiple values to the group of attributes but in different sets. In our example, for each course that a student has taken, she gets a separate set of values for the attributes (Course, Grade, Year). Thus the number of designed attributes she receives is comparable to the number of natural attributes she has; further the designed policy is comparable in size to that of a policy that did not enforce that attributes from different courses are not mixed together. In short, using CP-ASBE, we can obtain efficient ciphertext policy encryption schemes for several scenarios where existing CP-ABE scheme are insufficient.

Expressiveness in terms of Attribute-Databases Supported. Some of the flexibility illustrated above can be understood by viewing the association of attributes to a user as an entry in a database table. In such a table — which we will call the *attribute table* — each row stands for a user and each column (other than user identity) for an attribute.[1] The policy associated with a cipher-text could be considered a query into this table, to identify all users whose attributes satisfy a certain predicate.

The expressive power of a CP-ABE scheme is given by the class of queries into this table that the scheme can support. For instance, BSW CP-ABE [2] supports a large class of such queries. One challenge to increase the expressive power would be to broaden this class. However, there is another important dimension in which the expressive power of CP-ABE scheme can be improved, by supporting a more general class of attribute tables. The above description of CP-ABE required that each user ID appears in only one row in the table. (In other words, the user ID must be a "superkey" in the attribute

[1] In the case of a "large universe" of attributes, the number of columns could be very large — say all strings of 256 bits – and the resulting sparse table will never be stored directly as a table. Our examples shall mostly use the small universe scenarios, though they extend to the large universe setting as well.

table.) Of course, a table can be forced to have this property, but leading to large blow ups in the number of designed attributes that a user receives or the size of the designed policy. On the other hand, a CP-ASBE scheme can directly support a table with multiple rows per user: attributes in each row is given as a separate set.

2.2 Supporting Multiple Value Assignments

A major motivation for CP-ASBE is to support multiple value assignments for a given attribute in a single key.[2] To illustrate this, suppose score is a 6-bit integer representing the score a user receives in a game. (The user may possess several other attributes in the system.) The user can play the game several times and receive several values for score. This numerical attribute will be represented by 12 boolean attributes: score_bit0_0, score_bit0_1, . . ., score_bit6_0 and score_bit6_1, corresponding to the values 0 and 1 for the six bits in the binary representation of the value. Now consider a user who has two values of score, 33 (binary 100001) and 30 (binary 011110). By obtaining attributes for the bit values of these two numbers, the user gets all 12 boolean attributes, effectively allowing him to pretend to have any score he wants.

CP-ASBE solves this problem elegantly: each value assignment of the numerical attribute is represented in a separate set with six boolean attributes each (one for each bit position). Note that attributes other than score need not be repeated.

Application: Efficient revocation. ABE schemes suffer from lack of an effective revocation mechanism for keys that have been issued (just like IBE). To address this in CP-ABE in a limited manner, Bethencourt *et al.* [2] propose adding an expiration_time attribute to a user's key indicating the time (i.e., a numerical value) until which the key is considered to be valid. Then a policy can include a check on the expiration_time attribute as a numerical comparison. However, in practice the validity period of sensitive attributes has to kept small to reduce the *window of vulnerability* when a key is compromised, *e.g.* a day, a week or a month. At the end of this period the entire key will have to be re-generated and re-distributed with an updated expiration time imposing a heavy burden on the key server and key distribution process. In a practical implementation of a system using CP-ABE, the main efficiency bottle-neck in the system is the centralized key generation by an attribute authority [3].

CP-ASBE solves this problem more efficiently. First, we observe that while key validity is limited because of the window of vulnerability, the actual attribute assignments change far less frequently. Second, we obseve that it is possible to add attributes retroactively to a user key, both in BSW CP-ABE and CP-ASBE, if key server is able to maintain some state information about the user key. Then, by allowing multiple value assignments to the expiration_time attribute we can simply add a new expiration value to the existing key. Thus, while we require the key server to maintain some state we avoid the need to generate and distribute new keys on a frequent basis. This reduces the burden on the key server by a factor proportional to the average number of attributes in user keys.

[2] Note that multiple values for an attribute is relevant only when the attribute in question is not a boolean attribute (in a monotonic policy).

3 Related Work

While the concepts and ideas related to Attribute-Based Encryption have been alluded to in literature as far back as [5, 8] Sahai and Waters [18] proposed what is considered the first ABE scheme. Their scheme supported policies with a single threshold gate. Furthermore, the threshold value k, and size of the gate n used in a policy, are fixed during setup in their *Large Universe* construction. Pirretti *et al.*, [17] showed how to overcome this limitation of fixed k and n and demonstrated the use of threshold access policies for two applications. Traynor *et al.*, [20] further demonstrated its scalability by applying it to massive conditional access systems. Goyal *et al.*, [13] first defined the two complimentary forms of ABE, namely, KP-ABE and CP-ABE, and provided a construction for a KP-ABE[3] scheme. The proposed KP-ABE scheme supported all monotonic boolean encryption policies and was later extended by Ostrovsky *et al.*, [16] to support non-monotonic boolean formulas.

Bethencourt *et al.*, [2] gave the first construction for a CP-ABE scheme. Their construction supported all monotonic boolean encryption policies and the security of their scheme was argued in the generic group model. Cheung and Newport [7] gave the first standard model construction of CP-ABE scheme. While their scheme supported both positive and negative attributes it was limited to policies with single AND gates. Nishide *et al.*, [15] extended the scheme in [7] to support policy secrecy. Goyal *et al.* gave the first standard model construction of CP-ABE scheme that could support flexible policies [12]. Their scheme can realize all non-monotonic boolean formulas. However, since it is constructed using a KP-ABE scheme of [13], it is inefficient and has bounded ciphertext, *i.e.*, the size of supported policies is fixed at setup. Katz *et al.* proposed a KP-ABE scheme in [14] that can support flexible policies and achieve policy secrecy. This scheme can be used to realize CP-ABE schemes but such schemes have a bounded ciphertext. All the above ABE schemes are designed to work with one Attribute Authority (AA), a trusted entity that generates master parameters and distributes keys to users, and hence limited to a single domain. Chase extended [18] to multiple authorities in [6]. While most of the past work on CP-ABE schemes is focused on improving the expressibility of encryption policies and providing policy privacy ours is the first work to consider the flexibility of representing attributes in keys. All CP-ABE schemes to date can only support a monolithic set of user attributes which makes them inflexible and inefficient to capture naturally occurring "compound attributes". Our CP-ASBE scheme is the first to organize user attributes in keys and allow users to impose dynamic constraints on how attributes can be combined to satisfy policies, allowing our scheme more flexibility and efficiency when supporting "compound attributes".

Support for numerical attributes was first discussed in [2]. While the technique may be applicable to other schemes none of the existing CP-ABE schemes can support multiple value assignments for a given numerical attribute within a single key. Our CP-ASBE scheme is the first scheme to do so allowing it to support applications where such attribute assignments are needed without sacrificing flexibility of range queries (*i.e.*, numerical comparisons) in policies for those attributes.

[3] Since the scheme proposed in [18] supports policies with a single threshold gate it can be viewed either as a KP-ABE or as a CP-ABE scheme.

4 Preliminaries

Bilinear Maps Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic (multiplicative) groups of order p, where p is a prime. Let g_1 be a generator of $\mathbb{G}_1$, and g_2 be a generator of $\mathbb{G}_2$. Then $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map if it has the following properties:

1. Bilinearity: for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, h) \neq 1$.

Usually, $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$. $\mathbb{G}$ is called a bilinear group if the group operation and the bilinear map e are both efficiently computable.

Key Structure In CP-ABE schemes, an encryptor specifies an access structure for a ciphertext which is referred to as the ciphertext policy. Only users with secret keys whose associated attributes satisfy the access structure can decrypt the ciphertext. In CP-ABE schemes so far, a user's key can logically be thought of as a set of elements each of which corresponds to an associated attribute, such that only elements within a single set may be used to satisfy any given ciphertext policy (*i.e.* collusion resistance). In our scheme however, we use a recursive set based key structure where each element of the set is either a set itself (*i.e.* a key structure) or an element corresponding to an attribute. We define a notion of *depth* for this key structure, which is similar to the notion of depth for a tree, that limits this recursion. That is, for a key structure with depth 2, members of the set at depth 1 can either be attribute elements or sets but members of a set at depth 2 may only be attribute elements. The following is an example of a key structure of depth 2:

$$\Big\{\textit{CS-Department, Grad-Student}, \{\textit{Course101, TA}\}, \{\textit{Course525, Grad-Student}\}\Big\}$$

The depth of key structures that can be supported by our scheme is a system parameter that should be decided at the time of setup. That is, if the system is setup with a depth parameter of 5, keys of depth 5 or less can be supported. For ease of exposition, we will describe our scheme for key structures of depth 2. But we note that our construction is easily generalized to support keys of any depth d where d is fixed at setup. The key structure defines unique labels for sets in the key structure. For key structures of depth 2, just an index (arbitrarily assigned) of the set among sets at depth 2 is sufficient to uniquely identify the sets. Thus if there are m sets at depth 2 then an unique index i where $1 \leq i \leq m$ is (arbitrarily) assigned to each set. The set at depth 1 is referred to as set 0 or simply the outer set. If ψ represents a key structure then let ψ_i represent the ith set in ψ. Individual attributes inherit the label of the set they are contained in and are uniquely defined by the combination of their name and their inherited label. That is, while a given attribute might appear in multiple sets it can appear only once in any set. In the above example, the the outer set and $\{$*Course525, Grad-Student*$\}$ are assigned labels 0 and 2 respectively, and the two instances of the attribute *Grad-Student* are distinguished by the unique combination of their inherited set label and attribute name, (0, *Grad-Student*) and (2, *Grad-Student*), respectively. By default, a user may only use attribute elements within a set to satisfy a given ciphertext policy. That is, a user with the key structure from the above example may combine individual attributes either from the outer set (*i.e.*, $\{$*CS-Department, Grad-Student*$\}$) or from the set $\{$*Course101, TA*$\}$

or from the set {*Course525, Grad-Student*} to satisfy the policy associated with a given ciphertext but may not combine attributes across the sets. However, an encryptor may choose to allow combining attributes from multiple sets to satisfy the access structure by designating *translating nodes* in the access structure as explained below.

Access Structure We build on the access structure used in [2] which is a tree whose non-leaf nodes are threshold gates. Each non-leaf node of the tree is defined by its children and a threshold value. Let nc_x denote the number of children and k_x the threshold value of node x, then $0 < k_x \leq nc_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = nc_x$ it is an AND gate. The access tree also defines an ordering on the children of a node, *i.e.*, they are numbered from 1 to nc_x. For node x such a number is denoted by **index(**x**)**. Each leaf node y of the tree is associated with an attribute which is denoted by **att(**y**)**. Furthermore, the encrypting user may designate some nodes in an access tree as *translating nodes*. Their function will become clear as we discuss below the conditions under which a key structure is said to satisfy an access tree.

Let $\mathcal{T}$ be an access tree whose root node is r. Let $\mathcal{T}_x$ denote a subtree of $\mathcal{T}$ rooted at node x. Thus $\mathcal{T}_r$ is the same as $\mathcal{T}$. Now we will define the conditions under which a key structure ψ is said to satisfy a given access tree $\mathcal{T}$ assuming there are no designated translating nodes in the access tree. We will then extend the definition to consider the presence of translating nodes. A key structure ψ is said to satisfy the access tree $\mathcal{T}$ if and only if $\mathcal{T}(\psi)$ returns a non-empty set S of labels. We evaluate $\mathcal{T}_x(\psi)$ recursively as follows. If x is a non-leaf node we evaluate $\mathcal{T}_{x'}(\psi)$ for all children x' of x. $\mathcal{T}_x(\psi)$ returns a set S_x containing unique labels such that for every label $lbl \in S_x$ there exists at least one set of $k \geq k_x$ children such that for each child x' of these k children $S_{x'}$ contains the label lbl. If x is a leaf node then the set S_x returned by $\mathcal{T}_x(\psi)$ contains a label lbl if and only if $att(x) \in \psi_{lbl}$. Thus a key structure is is said to satisfy an access tree if it contains at least one set that has all the attributes needed to satisfy the access tree. Note that attributes belonging to multiple sets in the key structure cannot be combined to satisfy the access tree.

However, if there are designated translating nodes in the access tree, the algorithm $\mathcal{T}(\psi)$ is modified as follows. The algorithm $\mathcal{T}_{\S}(\psi)$ is the same as above when x is a leaf node. When x is a non-leaf node we evaluate $\mathcal{T}_{x'}(\psi)$ for all children x' of x. $\mathcal{T}_x(\psi)$ returns a set S_x containing unique labels such that for every label $lbl \in S_x$ there exists at least one set of $k \geq k_x$ children such that for each child x' of these k children $S_{x'}$ either contains the label lbl or x' is a translation node and $S_{x'} \neq \emptyset$. Thus, if node x is a designated translating node then, even if the attribute elements used to satisfy the predicate represented by the subtree rooted at x belong to a different set in the key structure than those used to satisfy the predicates represented by the siblings of x the decrypting user is able to combine them to satisfy the predicate represented by the parent node of x.

Syntax of CP-ASBE Scheme A CP-ASBE scheme consists of four algorithms, **Setup**, **KeyGen**, **Encrypt** and **Decrypt**. The algorithm **Setup** produces a master key and a public key for the scheme. **KeyGen** takes as input the master-key, a user's identity and an attribute set; it produces a secret key for the user. **Encrypt** takes as input the public key of the scheme, a message and an access tree, and outputs a ciphertext. Finally,

Decrypt takes a ciphertext and a secret-key (produced by **KeyGen**), and if the access-tree used to construct the ciphertext is satisfied by the attribute set for which the secret-key was generated, then it recovers the message from the ciphertext.

Security of CP-ASBE Scheme Our notion of *message indistinguishability* for CP-ASBE scheme against *chosen-plaintext attacks* is similar to that for CP-ABE schemes [2].

Setup. The challenger runs the Setup algorithm and gives public parameters, PK, to the adversary.

Phase 1. The adversary makes repeated queries for private keys corresponding to attribute sets $\mathbb{A}^1, \ldots, \mathbb{A}^{q_1}$.

Challenge. The adversary submits two equal length messages M_0 and M_1, and a challenge access structure $\mathcal{T}^*$ such that none of the private keys obtained in Phase 1 corresponding to attribute sets $\mathbb{A}^1, \ldots, \mathbb{A}^{q_1}$ satisfy the access structure. The challenger flips a random coin b, and encrypts M_b under $\mathcal{T}^*$. The resulting ciphertext CT is given to the adversary.

Phase 2. Phase 1 is repeated with the restriction that none of the attribute sets $\mathbb{A}^{q_1+1}, \ldots, \mathbb{A}^{q}$ satisfy the access structure corresponding to the challenge.

Guess. The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $Pr[b' = b] - \frac{1}{2}$. This game could easily be extended to include chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2.

Definition 1. *A CP-ASBE scheme is secure against chosen-plaintext attacks if all probabilistic polynomial time adversaries have at most a negligible advantage in the game above.*

5 Our CP-ASBE Construction

A key challenge in designing CP-ABE schemes is preventing users from pooling together their attributes. BSW CP-ABE achieves this by binding together all the attribute key components for each user with a random number unique to the user. Since in a CP-ASBE scheme one must prevent arbitrary combination of attributes belonging to different sets (even if they belong to the same user), a natural idea would be to similarly use a unique random number for binding together attribute key components for each set, in addition to using a random number for each user. However, a CP-ASBE scheme must also support *specific combinations* of attributes from different sets, as specified in an access-tree. The key idea in our construction is to include judiciously chosen additional values in the ciphertext (and in the key) that will allow a user to combine attributes from multiple sets all belonging to the same user. As it turns out, such a modification could introduce new subtle ways for multiple users to combine their attributes. Our construction shows how to thwart such attacks, using appropriate levels of randomization among different users' keys.

Let $\mathbb{G}_0$ be a bilinear group of prime order p and let g be a generator of $\mathbb{G}_0$. Let $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ denote a bilinear map. Let $H : \{0,1\}^* \rightarrow \mathbb{G}_0$ be a hash function

that maps any arbitrary string to a random group element. We will use this function to map attributes described as arbitrary strings to group elements.

Setup($d = 2$) The setup algorithm chooses random exponents $\alpha, \beta_i \in \mathbb{Z}_p \forall i \in \{1, 2\}$. The algorithm sets the public key and master key as:

$$\begin{aligned}\text{PK} =& (\mathbb{G}, g, h_1 = g^{\beta_1}, f_1 = g^{\frac{1}{\beta_1}}, h_2 = g^{\beta_2}, f_2 = g^{\frac{1}{\beta_2}}, e(g,g)^{\alpha}) \\ \text{MK} =& (\beta_1, \beta_2, g^{\alpha})\end{aligned}$$

Note that to support key structures of depth d, i will range from 1 to d.

KeyGen(MK, $\mathbb{A}$, u). Here u is the identity of a user and $\mathbb{A} = \{A_0, A_1, \ldots, A_m\}$ is a key structure. A_0 is the set of individual attributes in the outer set (*i.e.* set 0) and A_1 to A_m are sets of attributes at depth 2 that the user has. Let $A_i = \{a_{i,1}, \ldots, a_{i,n_i}\}$. That is, $a_{i,j}$ denotes the j-th attribute appearing in set A_i, and n_i denotes the number of attributes in the set A_i. (Note that for different values of (i, j), $a_{i,j}$ can be the same attribute.) The key generation algorithm chooses a unique random number, $r^{\{u\}} \in \mathbb{Z}_p$, for user u. It then chooses a set of m unique random numbers, $r_i^{\{u\}} \in \mathbb{Z}_p$, one for each set $A_i \in \mathbb{A}, 1 \leq i \leq m$. For set A_0, $r_0^{\{u\}}$ is set to be the same as $r^{\{u\}}$. It also chooses a set of unique random numbers, $r_{i,j}^{\{u\}} \in \mathbb{Z}_p$, one for each (i, j), $0 \leq i \leq m, 1 \leq j \leq n_i$. The issued key is:

$$\begin{aligned}\text{SK}_u = \Bigg(& \mathbb{A}, D = g^{\frac{(\alpha + r^{\{u\}})}{\beta_1}}, \\ & D_{i,j} = g^{r_i^{\{u\}}} \cdot H(a_{i,j})^{r_{i,j}^{\{u\}}}, D'_{i,j} = g^{r_{i,j}^{\{u\}}} \quad \text{for } 0 \leq i \leq m, 1 \leq j \leq n_i, \\ & E_i = g^{\frac{(r^{\{u\}} + r_i^{\{u\}})}{\beta_2}} \quad \text{for } 1 \leq i \leq m \Bigg)\end{aligned}$$

Elements E_i enable translation from $r_i^{\{u\}}$ (*i.e.*, set A_i) to $r^{\{u\}}$ (*i.e.*, the outer set A_0) at the translating nodes. Elements E_i and $E_{i'}$ can be combined as $E_i / E_{i'}$ to enable translation from $r_{i'}^{\{u\}}$ (*i.e.*, set $A_{i'}$) to $r_i^{\{u\}}$ (*i.e.*, the set A_i) at the translating nodes.

Encrypt(PK, M, $\mathcal{T}$) M is the message, $\mathcal{T}$ is an access tree. The algorithm associates a polynomial q_τ with each node τ (including the leaves) in the tree $\mathcal{T}$. These polynomials are chosen in the following way in a top-down manner, starting from the root node R. For each internal node τ in the tree, the degree d_τ of the polynomial q_τ is set to be one less than the threshold value k_τ of that node, that is, $d_\tau = k_\tau - 1$. For leaf nodes the the degree is set to be 0. For the root node R the algorithm picks a random $s \in Z_p$ and sets $q_R(0) = s$. Then, it chooses d_R other points randomly to define the polynomial q_R completely. For any other node τ, it sets $q_\tau(0) = q_{parent(\tau)}(index(\tau))$ and chooses d_τ other points randomly to completely define q_τ where **parent**(τ) denotes the parent node of τ. Let $\mathbb{Y}$ denote the set of leaf nodes in $\mathcal{T}$. Let $\mathbb{X}$ denote the set of *translating nodes* in the access tree $\mathcal{T}$. Then the ciphertext CT returned is as follows:

$$\begin{aligned}\text{CT} = \big(& \mathcal{T},\ \tilde{C} = M \cdot e(g,g)^{\alpha \cdot s}, C = h_1^s, \bar{C} = h_2^s,\ \forall y \in \mathbb{Y} : C_y = g^{q_y(0)}, \\ & C'_y = H(att(y))^{q_y(0)},\ \forall x \in \mathbb{X} : \hat{C}_x = h_2^{q_x(0)} \big)\end{aligned}$$

Translating values $\hat{C}'_x s$ together with $E_i{}' s$ in user keys allow translation between sets at a translating node x as will be described in the Decrypt function. Note that the element $\bar{C}$ is the same as $\hat{C}_r$ where r is the root node. A variant of the scheme would be where $\bar{C}$ is not included in the ciphertext but is only released at the discretion of the encrypting user as $\hat{C}_r$. This would restrict decrypting users to only use individual attributes in the outer set except when explicitly allowed by the encrypting user by designating translating nodes.

Decrypt(CT, SK_u) Here we describe the most straightforward decryption algorithm without regard to efficiency. The decryption algorithm is a recursive algorithm similar to the tree satisfaction algorithm described in Section 4. The decryption algorithm first runs the tree satisfaction algorithm on the access tree with the key structure *i.e.*, $\mathcal{T}(\mathbb{A})$, and stores the results of each of the recursive calls in the access tree $\mathcal{T}$. That is, each node t in the tree is associated with a set S_t of labels that was returned by $\mathcal{T}_t(\mathbb{A})$. If $\mathbb{A}$ does not satisfy the tree $\mathcal{T}$ then the decryption algorithm returns $\perp$. Otherwise the decryption algorithm picks one of the labels, i, from the set returned by $\mathcal{T}(\mathbb{A})$ and calls the a recursive function **DecryptNode**(CT, SK_u, t, i) on the root node of the tree. Here CT is the ciphertext $\text{CT} = (\mathcal{T}, \tilde{C}, C, \forall y \in Y : C_y, C'_y, \forall x \in X : \hat{C}_x)$, SK_u is a private key, which is associated with a key structure denoted by $\mathbb{A}$, t is a node from $\mathcal{T}$, and i is a label denoting a set of $\mathbb{A}$. Note that the ciphertext CT now contains tree information that is augmented by the results from $\mathcal{T}(\mathbb{A})$. **DecryptNode**(CT, SK_u, t, i) is defined as follows.

If $t \in \mathbb{Y}$, *i.e.*, node t is a leaf node, then **DecryptNode**(CT, SK_u, t, i) is defined as follows. If $att(t) \notin A_i$ where $A_i \in \mathbb{A}$ then $DecryptNode(\text{CT}, \text{SK}_u, t, i) = \perp$. If $att(t) = a_{i,j} \in A_i$ where $A_i \in \mathbb{A}$ then:

$$\begin{aligned} DecryptNode(\text{CT}, \text{SK}_u, t, i) =& \frac{e(D_{i,j}, C_t)}{e(D'_{i,j}, C'_t)} \\ =& \frac{e(g^{r_i^{\{u\}}} \cdot H(a_{i,j})^{r_{i,j}^{\{u\}}}, g^{q_t(0)})}{e(g^{r_{i,j}^{\{u\}}}, H(a_{i,j})^{q_t(0)})} = e(g,g)^{r_i^{\{u\}} \cdot q_t(0)} \end{aligned}$$

Note that set from which the satisfying attribute $a_{i,j}$ was picked is implicit in the result $e(g,g)^{r_i^{\{u\}} \cdot q_t(0)}$ (*i.e.*, indicated by $r_i^{\{u\}}$). When $t \notin \mathbb{Y}$, *i.e.*, node t is a non-leaf node, then $DecryptNode(\text{CT}, \text{SK}_u, t, i)$ proceeds as follows:

1. Compute $\mathbb{B}_t$ which is an arbitrary k_t sized set of child nodes z such that $z \in B_t$ only if either (1) label $i \in S_z$ or (2) label $i' \in S_z$ for some $i' \neq i$ and z is a translating node. If no such set exists then return $\perp$.
2. For each node $z \in B_t$ such that label $i \in S_z$ call $DecryptNode(\text{CT}, \text{SK}_u, t, i)$ and store output in F_z.
3. For each node $z \in B_t$ such that $(1, i') \in S_z$ and $i' \neq i$ call $DecryptNode(\text{CT}, \text{SK}_u, t, i')$ store output in F'_z. If $i \neq 0$ then translate F'_z to F_z as follows:

$$\begin{aligned} F_z &= e(\hat{C}_z, E_i / E_{i'}) \cdot F'_z \\ &= e(g^{\beta_2 \cdot q_z(0)}, g^{\frac{r_i^{\{u\}} - r_{i'}^{\{u\}}}{\beta_2}}) \cdot e(g,g)^{r_{i'}^{\{u\}} \cdot q_z(0)} = e(g,g)^{r^{\{u\}}{}_i \cdot q_z(0)} \end{aligned}$$

Otherwise, translate F'_z to F_z as follows:

$$F_z = \frac{e(\hat{C}_z, E_{i'})}{F'_z} = \frac{e(g^{\beta_2 \cdot q_z(0)}, g^{\frac{r^{\{u\}} + r_{i'}^{\{u\}}}{\beta_2}})}{e(g,g)^{r_{i'}^{\{u\}} \cdot q_z(0)}} = e(g,g)^{r^{\{u\}} \cdot q_z(0)}$$

4. Compute F_t using polynomial interpolation as follows:

$$F_t = \prod_{z \in B_t} F_z^{\Delta_{k,B'_z}(0)}, \qquad \text{where } k = index(z), B'_z = \{index(z) : z \in B_t\}$$
$$= \begin{cases} e(g,g)^{r_i^{\{u\}} \cdot q_t(0)} & \text{when } i \neq 0 \\ e(g,g)^{r^{\{u\}} \cdot q_t(0)} & \text{when } i = 0 \end{cases}$$

The output of $DecryptNode(\text{CT}, \text{SK}_u, r, i)$ function on the root node r is stored in F_r. If $i = 0$ we have $F_r = e(g,g)^{r^{\{u\}} \cdot q_r(0)} = e(g,g)^{r^{\{u\}} \cdot s}$ otherwise we have $F_r = e(g,g)^{r_i^{\{u\}} \cdot s}$. If $i \neq 0$ then we compute F as follows:

$$F = \frac{e(\hat{C}_r, E_i)}{F_r} = \frac{e(g^{\beta_2 \cdot q_r(0)}, g^{\frac{r^{\{u\}} + r_i^{\{u\}}}{\beta_2}})}{e(g,g)^{r_i^{\{u\}} \cdot q_r(0)}} = e(g,g)^{r^{\{u\}} \cdot q_r(0)} = e(g,g)^{r^{\{u\}} \cdot s}$$

Otherwise $F = F_r$. The decryption algorithm then computes following:

$$\frac{\tilde{C} \cdot F}{e(C, D)} = \frac{M \cdot e(g,g)^{\alpha \cdot s} \cdot e(g,g)^{r^{\{u\}} \cdot s}}{e(g^{s \cdot \beta_1}, g^{\frac{(r^{\{u\}} + \alpha)}{\beta_1}})} = M$$

Note how two elements E_i and $E_{i'}$ together with a translating value $\hat{C}_t$ at a node t were used to translate between sets i and i' at node t in step 3. Similarly, note how a single element E_i together with a translating value was used to translate between set i and the outer set. We note that if $\beta_1 = \beta_2$ then the scheme would become insecure as colluding users could transitively translate from inner set i to outer set and then from one key to the other by using the D elements from their keys. Thus we need a unique β for every level that we need to support. We emphasize that while we described our scheme for key structures of depth 2 it is easily generalized to key structures of arbitrary depth d which is fixed at setup.

5.1 Security

The security proof for our scheme closely follows that of BSW CP-ABE [2] and uses generic group [4, 19] and random oracle models [1]. We give the detailed proof in the full version [] of the paper but we state the theorem and provide some intuition here.

Generic Bilinear Group [4]. A generic group $\mathbb{G}_0$ with a bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ can be modeled by an oracle which uses random strings as handles for the elements in the two groups $\mathbb{G}_0$ and $\mathbb{G}_1$.[4] More precisely, we consider an oracle $\mathcal{O}$,

[4] We remark that it is not important to model the handles as *random* strings, but only as distinct handles that can be named by the adversary. But we stick to the convention from [4], that was used in [2], whose proof ours most closely resemble.

which picks two random encodings of the additive group $\mathbb{F}_p$ into sufficiently long strings, *i.e.*, injective maps $\psi_0, \psi_1 : \mathbb{F}_p \to \{0,1\}^m$, where $m > 3\log(p)$. We write $\mathbb{G}_0 = \{\psi_0(x) | x \in \mathbb{F}_p\}$ and $\mathbb{G}_1 = \{\psi_1(x) | x \in \mathbb{F}_p\}$. The oracle provides access to the group operations (which we shall refer to as multiplication) in either group: for example, queries of the form $(\text{multiply}_0, h, h')$ and $(\text{inverse}_0, h)$, will be answered respectively by $\psi_0(\psi_0^{-1}(h) + \psi_0^{-1}(h'))$, $\psi_0(-\psi_0^{-1}(h))$. If h or h' is not in the range of ψ_0, then the oracle returns $\perp$. The oracle also provides access to the identity elements $(\psi_0(0), \psi_1(0))$, and canonical generators $(\psi_0(1), \psi_1(1))$ in the two groups, as well as the ability to sample random elements in the groups. In addition, given a query (pair, h, h'), where $h = \psi_0(\alpha)$ and $h' = \psi_0(\beta)$, $\mathcal{O}$ returns $h'' = \psi_1(\alpha\beta)$. To relate to the notation of bilinear groups used in our construction, we will denote $\psi_0(1)$ by g and $\psi_0(x)$ by g^x. Similarly we will let $e(g,g)^y$ denote $\psi_1(y)$. Then the above pairing query to the oracle will be written as $e(g^\alpha, g^\beta)$ and the response as $e(g,g)^{\alpha\beta}$.

Finally, the oracle $\mathcal{O}$ also includes a random function $H : \{0,1\}^* \to \mathbb{G}_0$. It takes queries of the form (hash, a) for arbitrarily long strings a and returns $H(a)$.

Theorem 1. *Let $\mathcal{O}$, $\mathbb{G}_0$, $\mathbb{G}_1$, and H be as defined above. For any adversary $\mathcal{A}$ with access to $\mathcal{O}$ in the security game for the CP-ASBE scheme in Section 5 (using $\mathbb{G}_0$, $\mathbb{G}_1$, and H), suppose q is an upper-bound on the total number of group elements it receives from queries to $\mathcal{O}$ and interaction with the CP-ASBE security game. Then the advantage of $\mathcal{A}$ in the CP-ASBE security game is $O(q^2/p)$.*

Proof Intuition Let us say that s is the random secret split according to the access structure $\mathcal{T}$ as described in the **Encrypt** function of Section 5. Let $\mathcal{T}'$ be an access structure derived from $\mathcal{T}$ by removing the sub-trees under all translating nodes, *i.e.*, translating nodes become leaf nodes. For simplicity, let us assume for now [5] that all the leaves of $\mathcal{T}'$ are translating nodes in the original access structure $\mathcal{T}$. Let $q_t(0)$ represent the secret share associated with a translating node t. A user has to obtain $e(g,g)^{\alpha s}$ to recover the message encrypted using the access structure $\mathcal{T}$. He could pair $C = g^{\beta_1 s}$ given in the ciphertext with $D = g^{(\alpha + r^{\{u\}})/\beta_1}$ in his key to obtain $e(g,g)^{\alpha s + r^{\{u\}} s}$, *i.e.*, $e(g,g)^{\alpha s}$ blinded by $e(g,g)^{r^{\{u\}} s}$. A user can cancel out $e(g,g)^{r^{\{u\}} s}$ only if he satisfies the tree, *i.e.*, by obtaining a set of $e(g,g)^{r^{\{u\}} q_t(0)}$ that can reconstruct $e(g,g)^{r^{\{u\}} s}$. One can think of the key components given for each set of attributes in the key structure as a unique key under the BSW scheme. That is, if $r^{\{u\}}$ is the unique random number used in our CP-ASBE key then the set of key components (including the translation element) corresponding to each set A_i can be thought of as a BSW key issued using a master secret key $(\beta_2, g^{r^{\{u\}}})$. Furthermore, each of the sub-trees rooted at a translating node can be thought of an access structure under the BSW scheme. Thus a given sub-tree can only be satisfied using attributes from a single set, *i.e.* a single BSW key, as BSW is collusion resistant[6]. Thus a user who has a key with a set that can satisfy the sub-tree under a translating node t can obtain $e(g,g)^{r^{\{u\}} q_t(0)}$. And since $r^{\{u\}}$ is unique to a CP-ASBE key, only attributes from sets within a single CP-ASBE key can be used

[5] This assumption is not needed for the full proof.

[6] The proof in the Appendix makes it clear that the additional group elements that are available to an adversary in our scheme do not adversely affect this collusion resistance.

to satisfy $\mathcal{T}'$ and thus the original access structure. While we prove that our scheme is secure against *chosen-plaintext attacks* it can be extended to be secure against *chosen-ciphertext attacks* using Fujisaki-Okamoto transformation [10].

6 Evaluation

In this section we discuss the efficiency of CP-ASBE scheme instantiated with two-levels, describe its implementation and evaluate its performance overhead relative to BSW CP-ABE.

Efficiency It is straightforward to estimate the efficiency of our key generation and encryption algorithms. In terms of computation, our key generation algorithm requires two exponentiations for every attribute in the key issued to the user and two exponentiations for every set (including recursive sets for a scheme with levels > 2) in the key. In terms of key size, the private key contains two group elements per attribute and one group element per attribute set. Compared to BSW the additional key generation cost is two exponentiations for every attribute set in terms of computation and one group element per attribute set in terms of size. Encryption involves two exponentiations per leaf node in the tree and one exponentiation per translating node in the tree. The ciphertext contains two group elements per leaf node and one group element per translating node. Compared to BSW the additional cost is one exponentiation per translating node in terms of computation and one group element per translating node in terms of size. The cost of decrypting a given ciphertext however varies depending on the key used for decryption. Even for a given key there might be multiple ways to satisfy the associated access tree. The decrypt algorithm needs, 1) two pairings for every leaf node used to satisfy the tree, 2) one pairing for every translating node on the path from the leaf node used to the root and 3) one exponentiation for every node on the path from the leaf node to the root. However, by employing the optimization technique of flattening the recursive calls to *DecryptNode*, as described in BSW [2] albeit modified to accommodate translating nodes, we can reduce the cost to 1) two pairings and one exponentiation per leaf node used and 2) one pairing and one exponentiation per translating node on the path from a used leaf node to the root[7]. Compared to BSW the additional cost is one pairing and one exponentiation per translating node on the path from a used leaf node to the root. In a multi-level (level > 2) instantiation the overhead will be per translation rather than per translating node as multiple translations may be needed at a given translating node for such instantiations.

Implementation We have implemented a two-level CP-ASBE scheme as described in Section 5 with the following two differences, 1) decryption is optimized and 2) attribute elements of the set at depth 1 are not treated as singleton sets, *i.e.*, key structure is like the first example in Section 4. Both the above changes improve the efficiency and performance.

[7] The optimization technique is not described in detail due to space constraints but will be included in an extended version of the paper.

Our implementation leverages the *cpabe* toolkit[8] developed for BSW which uses Pairing-Based Cryptography library[9]. The interface for the *cpasbe* toolkit is similar to that of *cpabe* toolkit and is as follows:

cpasbe-setup Generates a public key and a master key.

cpasbe-keygen Given a master key, generates a private key for a given set of attributes; compiles numerical attributes into 'bag of bits' representation and treats the resulting attributes as a 'set'.

cpasbe-enc Given a public key, encrypts a file under a given access policy; numerical comparisons in the policy are represented by access sub-trees comprising 'bag of bits' representation of the numerical attribute with the root node of the sub-tree treated as a translating node.

cpasbe-dec Decrypts a file, given a private key.

The *cpasbe* toolkit is similar to *cpabe* toolkit in that it supports numerical attributes and range queries (*i.e.*, numerical comparisons) in access policies. However, unlike in *cpabe* toolkit, numerical attributes in *cpasbe* are treated as sets and thus *cpasbe* toolkit supports multiple numerical value assignments to a given attribute in a single private key. Thus a user with a private key generated using the following command cannot claim any score other than 33 and 30.

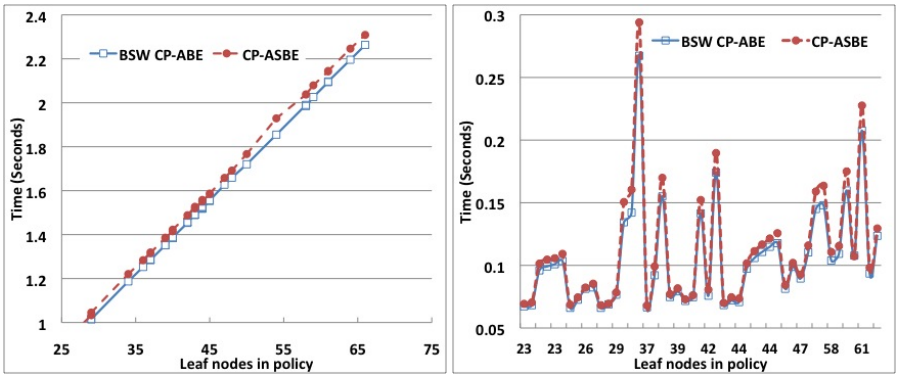
$ **cpasbe-keygen** -o tom-priv-key pub-key master-key 'score=33' 'score=30' tom

Performance Overhead A two-level CP-ASBE scheme provides better functionality over CP-ABE schemes in terms of, 1) better supporting compound attributes and 2) supporting multiple numerical value assignments for a given attribute in a single key. In order to gauge the cost of this additional functionality we compared the encryption, decryption and key generation times using randomly generated policies and associated keys with those of BSW CP-ABE scheme. The policies used to encrypt data were randomly generated formulae in the disjunctive normal form with leaf nodes ranging from 23 to 66. For each policy, a representative set of keys that satisfy the policy are generated and used for decryption. Specifically, 1) a key is generated for each conjunctive clause in the policy such that it satisfies the clause and 2) a key is generated for each combination of conjunctive clauses in the policy such that the key satisfies all the clauses in the combination. The generated keys had boolean attributes, ranging from 1 to 422, *i.e.*, including the "bag of bits" representation for numbers with 64 bits used to represent each integer. Decryption time for a policy is the average of decryption times with all the keys generated for that policy as described above. Experiments were run on a Linux box with quad core $3.0Ghz$ Intel Xeon and 2GB of RAM. Both implementations used a 160-bit elliptic curve group constructed on the curve $y^2 = x^3 + x$ over a 512-bit field.

While key generation time are not shown due to space constraints, as expected, they were found to be linear in the number of attributes in the key, and CP-ASBE imposed very little overhead over BSW CP-ABE. On an average, CP-ASBE imposed $18ms$ overhead per numerical attribute, *i.e.*, per set, in the key and no overhead when there are no numerical attributes. To put this overhead in perspective, generating a key with 2

[8] `http://acsc.csl.sri.com/cpabe/`

[9] `http://crypto.stanford.edu/pbc/`



(a) Encryption Time (b) Decryption Time

Fig. 1: Encryption and Decryption Times

numerical attributes (and 145 boolean attributes in total) took $5s$ seconds when using BSW CP-ABE scheme and $5.035s$ when using CP-ASBE scheme. Encryption and decryption times are shown in Figure 1. Encryption time is, as expected, linear in the number of leaves in the policy tree, and CP-ASBE imposed very little overhead when compared to BSW CP-ABE. On an average, CP-ASBE imposed $8.3ms$ overhead per translating node in the policy. Decryption time, however, varied significantly depending on the structure of the policy tree and the key used to decrypt. However, in this case too CP-ASBE scheme imposed very little to no overhead over BSW CP-ABE, $6.7ms$ on average. Overhead results are consistent with our efficiency analysis and performance numbers in general are consistent with those reported in [2].

7 Conclusion and Future Work

In this work we proposed CP-ASBE a form of CP-ABE that organizes user attributes into a recursive family of sets and allows users to impose dynamic constraints on how attributes may be combined. We demonstrated how CP-ASBE can naturally support compound attributes, and numerical attributes with multiple value assignments. We showed that it achieves this versatility with very little overhead through efficiency analysis and performance evaluation of a prototype implementation. An interesting direction for future research is to study the potential of CP-ASBE schemes and ABE schemes in general in supporting constructs similar to "OR roles" [11] and constraints like "dynamic mutually exclusive roles" that are common in traditional mediated RBAC settings. Other directions for future work are the design of efficient CP-ASBE schemes that are secure in the standard model and extending CP-ASBE to a multi-authority setting.

Acknowledgments

We would like to thank the reviewers for their valuable feedback. This work was supported by National Science Foundation under Grant No. CNS 07-16626 and by Office of

Naval Research under Grant No. N00014-07-1-1173. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

References

1. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
2. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, 2007.
3. R. Bobba, O. Fatemieh, F. Khan, A. Khan, C. A. Gunter, H. Khurana, and M. Prabhakaran. Attribute-Based Messaging: Access Control and Confidentiality. Under Revision for ACM Transactions on Information and System Security.
4. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In Cramer [9], pages 440–456.
5. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. *Advances in Cryptology-Crypto 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, 2001.
6. M. Chase. Multi-authority attribute based encryption. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534. Springer, 2007.
7. L. Cheung and C. Newport. Provably secure ciphertext policy abe. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 456–465, New York, NY, USA, 2007. ACM.
8. C. Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.
9. R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
10. E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.
11. L. Giuri. A New Model for Role-Based Access Control. In *Annual Computer Security Application Conference*, pages 249–255, December 1995.
12. V. Goyal, A. J. 0002, O. Pandey, and A. Sahai. Bounded Ciphertext Policy Attribute Based Encryption. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 579–591. Springer, 2008.
13. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.
14. J. Katz, A. Sahai, and B. Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.
15. T. Nishide, K. Yoneyama, and K. Ohta. Attribute-based encryption with partially hidden encryptor-specified access structures. In S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 111–129, 2008.

16. R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security*, 2007.
17. M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure attribute-based systems. In *ACM Conference on Computer and Communications Security*, pages 99–112, 2006.
18. A. Sahai and B. Waters. Fuzzy identity-based encryption. In Cramer [9], pages 457–473.
19. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.
20. P. Traynor, K. Butler, W. Enck, and P. McDaniel. Realizing massive-scale conditional access systems through attribute-based cryptosystems. In *Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS)*, February 2008.